Timely Notes and Supplementary Information for C-terp Version L3 2.22 (Lattice 3.0 Variant)

New 2.22 Features

New Main Menu Commands

Batch

Batch allows use of the batch facility from the main menu. Heretofore, the Batch facility was only available from the DOS command line (see Section "Batch Mode" in the chapter entitled "THE COMMAND LINE").

The Batch Command will prompt for the name of a file, a so-called batch file, that contains a sequence of C-terp commands. If an extent is not provided, ".ct" is assumed. A batch file may contain any sequence of commands except a batch command. If a "U *" (Unload all) command is issued from the batch, the batch file itself is not unloaded. Thus the batch file:

```
u  *
l  alpha1
l  alpha2
l  alpha3
```

serves to unload every file except the batch file and then proceeds to load the three files alpha1.c, alpha2.c and alpha3.c.

Debug

Debug operates just like Run (see the Run command in the the chapter entitled "THE MAIN MENU") except that it starts the program off in debug mode (see section "Interactive Debugging" in the chapter "DEBUGGING"). It is as if the program contained a call to breakpt() as the first executable statement in main(). As with Run, Debug will compile all modules that need to be compiled and will prompt for a command line before placing the program into execution.

The major reason for the inclusion of the Debug command is to avoid the necessity of using breakpt() so frequently. Once Interactive Debugging is entered, breakpoints can be set based on line number (with the <integer>c command) so that, in principle, breakpt() need not be used at all.

Dual Display Option

C-terp now supports the use of two screens where one screen is controlled through the monochrome adaptor and the other through the color graphics adaptor. Other combinations are possible through a judicious modification of of the module scr2save.c described below.

When this option is in effect, C-terp should be started with the system in monochrome mode (mode mono). All editing is done in that mode. Just before control is given to a user program, the mode is switched to color 80X25. Of course, the user program may set the mode to graphics in the usual manner.

During interactive debugging the program window and the command window are displayed on the monochrome screen. At this time, called functions may display information on the graphics screen

(using the 'D' (Display) command -- note the Capital 'D'). Using 'N' (Next step) and 'S' (Side step), you may step through the program with program output being displayed on the graphics screen.

To set up the dual display option you must compile and link into C-terp a small program called scr2save.c, provided on the distribution diskette. This module will take the place of scrsave.c, which is also found on the distribution diskette and described in the chapter "ADVANCED CONFIGURATION", section "Saving the Graphics Screen"). Please follow the compiling and linking procedures described in that section.

Source for scr2save.c (and scrsave.c) is provided so that users will be able to customize C-terp for other less common screens or screen combinations.

New error messages

A new error message (50) replaces a previous internal error (225):

50 Bad left operand -- The cursor is positioned at or just beyond either an "->" or a "." operator. These operators expect an expression primary on their left. Please enclose any complex expression in this position within parentheses.

Unload *

If an argument of * is provided to the main menu Unload command, each of the loaded files will be Unloaded. Unloading implies prompting for each file that had been modified since it was last written.

---
setjmp, longjmp
---

setjmp, longjmp are now supported. The description is as follows:

setjmp -- establish a 'platform' for longjmp

Synopsis:

```
#include "setjmp.h"

int setjmp( p )
    jmp_buf  p;
```

Description:

The setjmp function works in conjunction with longjmp to enable the user to 'escape out' of a nested function call sequence; i.e. to make a non-local goto. This can be of value, for example, if an error is discovered deep within a program's calling sequence and it is needed to jump back out to, say, the main program and avoid the return through the chain of calls.

A call upon setjmp(p) will save information in a buffer pointed to by p (p is really a pointer even though the type 'jmp_buf' defined within setjmp.h serves to disguise this fact). The companion function longjmp(p,k) will use the information from the same buffer to restore the state of the computation to that which it had been when setjmp was called. The only difference is that setexit will apparently return k. For example, a main program may resemble:

```
#include "setjmp.h"
jmp_buf buf;

main()
    {
    if( setjmp(buf) == 0 )
        {
        /* main program as usual. calls
```

```
            f() or somebody that calls f().
         */
        }
else
        {
        /* Control flows here if longjmp()
           is called. Do error processing.
         */
        }

    f(n)
        {
        if( n < 0 ) longjmp(buf,1);
        /* process the rest of f() */
        }
```

In the example, if the argument to f() is negative, flow goes back to the main program, returning a 1 from setjmp and the else condition is taken.

You may use the setjmp.h that comes with your compiler as the buffer defined there is large enough for C-terp. That way you only have one setjmp.h to worry about.

Serious disruption in function is possible if the following rule is not strictly obeyed. If the function which calls setexit ever itself returns, the saved environment is no longer valid and should not be used. This rule must be obeyed in a compiled environment as well as an interpreted environment.

longjmp -- jump to an environment established by setjmp.

Synopsis:

```
    #include "setjmp.h"

    int longjmp( p, k )
        jmp_buf  p;
        int k;
```

Description:

The longjmp function is used in conjunction with setjmp to make a non-local goto (see the description of setjmp for an example). Control goes to the environment established by setjmp() and setjmp() apparently returns the value k.

If the value provided for k is 0 it is changed to 1. This is for compatibility with many compilers and to protect the programmer from inadvertently passing a 0 back to the target environment.

You may use the setjmp.h that comes with your compiler as the buffer defined there is large enough for C-terp. That way you only have one setjmp.h to worry about.

New 2.21 Features

Pointers to internal functions have (at last) been made fully compatible with pointers to external functions. This means, for example, that a pointer to an interpreted function can be passed to an external routine which can then call the interpreted function. This can be quite useful when invoking external sort routines or interrupt establishing routines that accept pointers to functions as arguments. Also, many libraries such as Windows for Data (from Vermont Creative) or Panel (Roundhill Computer) require such pointers to establish help and/or validation. This could previously only be done via a circumlocution as outlined in the manual (Chapter 'LINKING TO EXTERNALS', Section 'Restrictions and Concerns', Subsection 'Pointers to Functions').

New 2.20 Features

- Virtual Memory Option. This is described in chapter "THE COMMAND LINE", under section "Options".

- Call-in: a facility to allow external functions to make calls into interpreted routines. This is described in chapter "LINKING TO EXTERNALS", under its own section.

- Capturing Trace Output. Trace output can now be diverted to a file. See chapter "THE MAIN MENU", section "Options".

- F7: the Previous File key. This key is a companion to F8 (Next File key) and is described in the chapter on "THE EDITOR".

- Rationalized external function pointers. We now let you pass pointers-to-external-functions to external functions (meaningfully). Coupled with the Call-in facility, this permits you to ultimately interpret the function. See section "Restrictions and Concerns" in chapter "LINKING TO EXTERNALS".

- Graphics debugging supported. By default we now save and restore the graphics screen for the standard CGS adaptor (if in graphics mode) and, as before, allow you to tailor the system for your own screen. See chapter "DEBUGGING", section "Interactive Debugging".

In addition there have been a number of small bugs fixed, and an improved install facility.

Version 2.20 Manual Modifications

Use FILES = ...

C-terp will not work properly unless you have configured your system to contain more files than the default value (8). Symptoms of this are C-terp's refusal to open files. The reason that '8 is not enough' is that the Lattice 3.0 standard library employs five at startup time and several are used internally by C-terp. Fortunately the fix is easy.

Please check the file CONFIG.SYS in your boot directory and make sure it contains the line:

FILES = #

where # is a number equal to 16 or higher. For example:

FILES = 16

INCLUDE environment variable

We are now supporting the INCLUDE environment variable, which is now becoming something of a standard. Its value is a sequence of directories separated by semi-colons (just like the PATH environment variable). It's effect is to cause searching for include files to be attempted in these directories if they are not found in the current directory. For C-terp the INCLUDE variable becomes the default setting of the Path Option within the Option sub-menu. It may be modified via options.

Note that INCLUDE, like any other environment variable is set with a SET command at the DOS

level.

Declaring malloc() and calloc()

A common problem for beginners (and there is no adequate warning about this in the manual) is that functions that return pointers such as malloc() and calloc() must be declared as in:

```
char *malloc();
```

If no such declaration is provided it is assumed the function returns an int (this is in accordance with C rules) and the result is that the returning pointer is truncated to 16 bits. This will result in an Error 107 message when you go to use the pointer (when writing into the area).

Control Break

We neglect to mention anywhere in the manual that a program can be interrupted with a Control Break (depressing the Ctrl key while pressing Break). Typing C will continue with the execution.

Easy Access to External Data

In the chapter entitled "LINKING TO EXTERNALS" subsection "Considerations in Accessing Data" we have you going through a clumsy four-edit process merely to reference one external data item. If you had to reference many items, this could indeed become tortuous.

For most (as far as I know all) compilers, this really isn't necessary. The external data object can be treated as just another external function as far as tblxn.c is concerned. C-terp will

eventually need to know its type but this can be picked up from some interpreted module that refers to the external data object. Therefore, just add the name of the external data object to the list of external functions that are passed into augtblxn and Presto, your linkage will be made. (See "Augmenting tblxn" later in the same chapter.)

PALLOC flag

The flag, PALLOC, #defined in tblxn.h (and fully described there), was inadvertently omitted from the manual. It should have been described in the chapter entitled LINKING TO EXTERNALS, section "Pointer Checking Flags". You may add the following description there.

PALLOC

indicates that the associated function returns a Pointer to ALLOCated region. This flag implies that the external is a function that returns a pointer to an area that is write-accessible by the calling program. The size of the area is determined by the declaration in the caller's program. Thus, if a function f() is declared pointer to struct and marked PALLOC then the size of the area is considered to be the size of the struct. fopen in some systems MUST be declared this way.

---
bl_alloc and bl_free
---

bl_alloc and bl_free can be used to resolve some storage allocation problems offered by some libraries. Most users will have NO NEED for these functions. They are NOT intended to be called from interpreted routines but rather from compiled

routines. It is expected that the greatest use will be by library venders in order to make their products more transparent to users of C-terp.

The problem: Some libraries allocate memory for their users and provide pointers which are then employed by the user to access and write into these areas. Since the allocation is done within the library, C-terp has no way of knowing that this storage is user-writable.

A call to bl_alloc(p,n) where p is a pointer and n is an integer indicating the size of an allocated region beginning with p indicates to C-terp that this storage area is writable by the program (for the duration of the run). Note that bl_alloc() does not allocate the storage.

A call to bl_free(p) is the inverse of bl_alloc(). It indicates that the pointer p (previously supplied to bl_alloc()) should no longer be considered as pointing into allocatable storage. Note that bl_free() does not free the storage.